دانشکده مهندسی کامپیوتر

ساختمان داده

تمرین ۱۰*

مبین داریوش همدانی
بابک بهکام کیا
سید صالح اعتمادی

نیم‌سال اول ۱۴۰۱-۱۴۰۰

ایمیل/تیمز	m_dariushhamedani@comp.iust.ac.ir babak_behkamkia@comp.iust.ac.ir
نام شاخه	fb_A10
نام پروژه/پوشه/پول ریکوست	A10
مهلت تحویل	۱۴۰۰/۹/۲۰

*تشکر ویژه از خانم مریم سادات هاشمی که در نیم‌سال اول سال تحصیلی ۹۷-۹۸ نسخه اول این مجموعه تمرین‌ها را تهیه فرمودند. همچنین از اساتید حل‌تمرین نیم‌سال اول سال تحصیلی ۹۹-۹۸ سارا کدیری، محمد مهدی عبدالله‌پور، مهدی مقدمی، مهسا قادران، علیرضا مرادی، پریسا یل‌سوار، غزاله محمودی و محمدجواد میرشکاری که مستند این مجموعه تمرین‌ها را بهبود بخشیدند، متشکرم.

توضیحات کلی تمرین

۱. ابتدا مانند تمرین های قبل، یک پروژه به نام A10 بسازید. همچنین پروژه تست متناظر آن را ساخته و مطابق راهنمای تمرین یک فایل‌ها را در پوشه متناظر اضافه کرده و تنظیمات مربوط به کپی کردن TestData به پوشه خروجی را در تنظیمات پروژه تست قرار دهید. دقت کنید که پروژه TestCommon فقط یکبار باید در ریشه گیت موجود باشد و نباید در هر تمرین مجدد کپی شود. برای روش ارجاع به این پروژه به تمرین شماره یک مراجعه کنید.

۲. کلاس هر سوال را به پروژه‌ی خود اضافه کنید و در قسمت مربوطه کد خود را بنویسید. هر کلاس شامل دو متد اصلی است:

- متد اول: تابع Solve است که شما باید الگوریتم خود را برای حل سوال در این متد پیاده سازی کنید.
- متد دوم: تابع Process است که مانند تمرین های قبلی در TestCommon پیاده سازی شده است. بنابراین با خیال راحت سوال را حل کنید و نگران تابع Process نباشید! زیرا تمامی پیاده سازی ها برای شما انجام شده است و نیازی نیست که شما کدی برای آن بزنید.

۳. اگر برای حل سوالی نیاز به تابع های کمکی دارید؛ می توانید در کلاس مربوط به همان سوال تابع تان را اضافه کنید.

توجه:
برای اینکه تست شما از بهینه سازی کامپایلر دات نت حداکثر بهره را ببرد زمان تست ها را روی بیلد **Release** امتحان کنید، درغیر اینصورت ممکن است تست های شما در زمان داده شده پاس نشوند.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using A10;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestCommon;

namespace A10.Tests
{
    [DeploymentItem("TestData")]
    [TestClass()]
    public class GradedTests
    {
        [TestMethod(), Timeout(1000)]
        public void SolveTest_Q1PhoneBook()
        {
            RunTest(new Q1PhoneBook("TD1"));
        }

        [TestMethod(), Timeout(1000)]
        public void SolveTest_Q2HashingWithChain()
        {
            RunTest(new Q2HashingWithChain("TD2"));
```

```
        }


        [TestMethod(), Timeout(1000)]
        public void SolveTest_Q3RabinKarp()
        {
            RunTest(new Q3RabinKarp("TD3"));
        }


        public static void RunTest(Processor p)
        {
            TestTools.RunLocalTest("A10", p.Process, p.TestDataName,
                p.Verifier, VerifyResultWithoutOrder: p.VerifyResultWithoutOrder,
                excludedTestCases: p.ExcludedTestCases);
        }


        /// <summary>
        /// This test is just to help you test your
        /// PreComputeHashes function. It is not graded
        /// </summary>
        [TestMethod()]
        public void PreComputeHashesTest()
        {
            // Uncomment the following line if you want to have it run
            // Assert.Inconclusive();
            string testStr = "aaaa";
            int patternLen = 2;
            long[] H = Q3RabinKarp.PreComputeHashes(
                testStr, patternLen, 101, 3);

            for (int i = 0; i < testStr.Length - patternLen + 1; i++)
            {
                long expectedHash =
                    Q2HashingWithChain.PolyHash(testStr, i, patternLen, 101, 3);
                Assert.AreEqual(expectedHash, H[i]);
            }
        }
    }
}
```

۱ Phone book

در این سوال شما باید یک دفترچه تلفن ساده را پیاده سازی کنید. برنامه ی شما باید بتواند درخواست های کاربر را پردازش کند. این درخواست ها به صورت زیر است:

- add number name

این بدان معنی است که کاربر یک فرد را با نام و شماره تلفنی که وارد کرده است را به دفترچه اضافه می کند. اگر فردی با چنین شماره ای در دفترچه تلفن وجود دارد، شما باید نام متناظر را بازنویسی کند.

- del number

این بدان معنی است که باید یک فرد با شماره تلفن ورودی را از دفترچه تلفن پاک کنید. اگر چنین شخصی وجود نداشته باشد، این درخواست را نادیده بگیرید.

- find number

این بدان معنی است که کاربر به دنبال فردی با شماره تلفن ورودی است. شما باید نام شخص متناظر با شماره ی تلفن را برگردانید.اگر چنین شخصی در دفترچه تلفن وجود نداشت، عبارت not found را برگردانید.

در هر خط از فایل ورودی یکی از درخواست های بالا وجود دارد. در هر خط از خروجی هم پاسخ به درخواست find قرار دارد.

ورودی نمونه	خروجی نمونه
12 add 911 police add 76213 Mom add 17239 Bob find 76213 find 910 find 911 del 910 del 911 find 911 find 76213 add 76213 daddy find 76213	Mom not found police not found Mom daddy

توجه کنید که برای راحتی شما کلاس contact را پیاده سازی کردیم. این کلاس دو ویژگی name و number دارد. شما می توانید در صورت نیاز از این کلاس استفاده کنید و یا آن را تغییر دهید. در این سوال شما باید تابع های Add و Delete و Find که در فایل PhoneBook.cs قرار دارد را پیاده سازی و کامل کنید.

در این سوال پیاده سازی ساده و ابتدایی برای شما انجام شده است که اگر تست کنید خواهید دید که نسبت به الگوریتم اصلی که شما باید پیاده سازی کنید بسیار کند است و بیش از اندازه طول می کشد.

```cs
using System;
using System.Linq;
using System.Collections.Generic;
using TestCommon;

namespace A10
{
    public class Contact
    {
        public string Name;
        public int Number;

        public Contact(string name, int number)
        {
            Name = name;
            Number = number;
        }
    }

    public class Q1PhoneBook : Processor
    {
        public Q1PhoneBook(string testDataName) : base(testDataName) { }

        public override string Process(string inStr) =>
            TestTools.Process(inStr, (Func<string[], string[]>)Solve);

        protected List<Contact> PhoneBookList;

        public string[] Solve(string [] commands)
        {
            PhoneBookList = new List<Contact>(commands.Length);
            List<string> result = new List<string>();
            foreach(var cmd in commands)
            {
                var toks = cmd.Split();
                var cmdType = toks[0];
                var args = toks.Skip(1).ToArray();
                int number = int.Parse(args[0]);
                switch (cmdType)
                {
                    case "add":
                        Add(args[1], number);
                        break;
                    case "del":
                        Delete(number);
                        break;
                    case "find":
                        result.Add(Find(number));
                        break;
                }
            }
            return result.ToArray();
        }

        public void Add(string name, int number)
        {
            for(int i=0; i<PhoneBookList.Count; i++)
```

```
            {
                if (PhoneBookList[i].Number == number)
                {
                    PhoneBookList[i].Name = name;
                    return;
                }
            }
            PhoneBookList.Add(new Contact(name, number));
        }

        public string Find(int number)
        {
            for (int i = 0; i < PhoneBookList.Count; i++)
            {
                if (PhoneBookList[i].Number == number)
                    return PhoneBookList[i].Name;
            }
            return "not found";
        }

        public void Delete(int number)
        {
            for (int i = 0; i < PhoneBookList.Count; i++)
            {
                if (PhoneBookList[i].Number == number)
                {
                    PhoneBookList.RemoveAt(i);
                    return;
                }
            }
        }
    }
}
```

۲ Hashing with chains

در این سوال شما باید یک جدول hash را به صورت زنجیره ای پیاده سازی کنید. Chaining یکی از رایج ترین روش های اجرای جدول های hash است. از چنین جدول hash ای می توان برای پیاده سازی یک دفترچه تلفن بر روی تلفن همراه یا ذخیره جدول رمز عبور کامپیوتر و سرویس وب استفاده کرد.

فرض کنید که تعداد bucket ها m باشد بنابراین برای پیاده سازی جدول hash به صورت زنجیره ای از تابع چند جمله ای زیر به عنوان تابع hash استفاده کنید.

$$h(S) = \left(\sum_{i=0}^{|S|-1} S[i]x^i \bmod p \right) \bmod m,$$

که در این تابع $s[i]$ کد ASCII ،المان i رشته یs است و

$$\mathrm{p} = 1\ 000\ 000\ 007\ ,\ \mathrm{x} = 263$$

برنامه شما باید دستور های زیر را پشتیبانی کند:

- add string این بدان معنی است که رشته را به جدول وارد کنید. اگر قبلا چنین رشته ای در جدول hash وجود دارد،این درخواست را نادیده بگیرید.
- del string

  این بدان معنی است که رشته را از جدول حذف کنید. اگر چنین رشته ای در جدول hash وجود ندارد، پس درخواست را نادیده بگیرید.
- find string

  این بدان معنی است که بسته به اینکه آیا جدول شامل رشته هست یا نه، خروجی "yes" یا "no" را برگردانید .
- check i

  این بدان معنی است که محتوای i امین لیست در جدول را خروجی بدهید. با استفاده از space المان ها از هم جدا می شوند. اگر لیست i ام خالی باشد، یک خط خالی را در خروجی نمایش دهید.

هنگام وارد کردن یک رشته جدید به زنجیره hash باید آن را در ابتدای زنجیره وارد کنید. خط اول ورودی تعداد Bucket هاست و هر یک از خطوط بعدی یکی از درخواست های بالا می باشد. هر یک از خطوط خروجی جواب درخواست های check و find به ترتیبی که این دستور ها در ورودی آمده است، می باشد.

ورودی نمونه	خروجی نمونه
5 12 add world add HellO check 4 find World find world del world check 4 del HellO add luck add GooD check 2 del good	HellO world no yes HellO GooD luck

در این سوال شما باید تابع های Add و Delete و Find و check که در فایل HashingWithChain.cs قرار دارد را پیاده سازی و کامل کنید. دقت کنید برخی از مواردی که برای حل این سوال نیاز دارید برای شما پیاده سازی شده است.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using TestCommon;
5
6  namespace A10
```

```
{
    public class Q2HashingWithChain : Processor
    {
        public Q2HashingWithChain(string testDataName) : base(testDataName) { }

        public override string Process(string inStr) =>
            TestTools.Process(inStr, (Func<long, string[], string[]>)Solve);


        public string[] Solve(long bucketCount, string[] commands)
        {
            List<string> result = new List<string>();
            foreach (var cmd in commands)
            {
                var toks = cmd.Split();
                var cmdType = toks[0];
                var arg = toks[1];

                switch (cmdType)
                {
                    case "add":
                        Add(arg);
                        break;
                    case "del":
                        Delete(arg);
                        break;
                    case "find":
                        result.Add(Find(arg));
                        break;
                    case "check":
                        result.Add(Check(int.Parse(arg)));
                        break;
                }
            }
            return result.ToArray();
        }

        public const long BigPrimeNumber = 1000000007;
        public const long ChosenX = 263;

        public static long PolyHash(
            string str, int start, int count,
            long p = BigPrimeNumber, long x = ChosenX)
        {
            long hash = 0;
            return hash;
        }

        public void Add(string str)
        {
        }

        public string Find(string str)
        {
            return "";
        }

```

```
        public void Delete(string str)
        {
        }

        public string Check(int i)
        {
            return "";
        }
    }
}
```

۳ Find pattern in text

در این سوال شما باید الگوریتم Rabin-Karp را برای جستجوی یک الگو در یک متن پیاده سازی کنید. در فایل ورودی دو رشته وجود دارد که رشته ی اول الگو و رشته ی دوم متن می باشد. در خروجی هم باید مکان هایی از متن که در آن الگو داده شده اتفاق افتاده است را برگردانید. یعنی:

ورودی نمونه	خروجی نمونه
aba abacaba	0 4

همان طور که ملاحظه می کنید الگو ی aba در متن داده شده، یک بار در مکان صفر اتفاق افتاده است و بار دیگر در مکان ۴. بنابراین خروجی به صورت زیر خواهد بود:

ورودی نمونه	خروجی نمونه
Test testTesttesT	4

ورودی نمونه	خروجی نمونه
aaaaa baaaaaaa	1 2 3

همانطور که در تکه کد زیر مشاهده می کنید، در این سوال پیاده سازی ساده و ابتدایی برای شما انجام شده است. اگر این برنامه را تست کنید خواهید دید که نسبت به الگوریتم اصلی که شما باید پیاده سازی کنید بسیار کند است و بیش از اندازه طول می کشد.

```
using System;
using System.Collections.Generic;
using TestCommon;

```

```
namespace A10
{
    public class Q3RabinKarp : Processor
    {
        public Q3RabinKarp(string testDataName) : base(testDataName) { }

        public override string Process(string inStr) =>
            TestTools.Process(inStr, (Func<string, string, long[]>)Solve);

        public long[] Solve(string pattern, string text)
        {
            List<long> occurrences = new List<long>();
            int startIdx = 0;
            int foundIdx = 0;
            while ((foundIdx = text.IndexOf(pattern, startIdx)) >= startIdx)
            {
                startIdx = foundIdx + 1;
                occurrences.Add(foundIdx);
            }
            return occurrences.ToArray();
        }


        public static long[] PreComputeHashes(
            string T,
            int P,
            long p,
            long x)
        {
            return new long[] { };
        }
    }
}
```

۴ Bloom filter[۱]

$BloomFilter$ یک ساختمان داده فشرده بر اساس $HashFunction$ است. این ساختمان داده شبیه $HashTable$ است با چند تفاوت: اول اینکه برخلاف $HashTable$ این ساختمان داده به ازای کلید، مقداری بر نمیگرداند. تنها کاری که می کند این است که می گوید آیا این کلید قبلا اضافه شده یا نه. در واقع فقط دو متد دارد $Add(key)$ و $Test(key)$، تفاوت دوم اینکه فقط می شود به این ساختمان داده ای کلید اضافه کرد(نمی توانید حذف کنید). و مهمترین تفاوت اینکه اگر جواب $Test$ منفی باشد، جواب صد در صد درست است ولی اگر جواب مثبت باشد، یک احتمالی هم وجود دارد که واقعا درست نبوده باشد. به اینکه چند درصد از جواب های مثبت تست واقعا درست بوده اصطلاحا $FalsePositiveRate$ گفته می شود. این مقدار بستگی به دو عامل دارد. یکی اندازه این ساختمان داده(اندازه ساختمان داده بر اساس بیت است و بعد از اضافه کردن اولین کلید، دیگر قابل تغییر نیست). دوم تعداد و نوع تابع های $hash$ استفاده شده. هر چه اندازه این ساختمان داده بزرگتر باشد درصد $FalsePositiveRate$ کمتر میشود. تابع های $hash$ باید از هم مستقل یا مقدارشان به هم مربوط نباشد. بطوریکه اگر دو کلید متفاوت با یک تابع $hash$ مقدار یکسان

[۱] این سوال امتیازی است.

داشتند، با $hash$ دوم حتی المکان مقدار یکسان نداشته باشند. بهتر است تعداد توابع هم تقریبا 0.7 نسبت به تعداد کل کلید ها به تعداد بیت ها باشد. مستقل از اندازه کلید، تقریبا ده بیت به ازای هر کلید منجر به $FalsePositiveRate$ حدود ۱ درصد میشود. مثلا هزار تا کلید، ده هزار بیت و هفت تا تابع $hash$ داشته باشیم(البته اینها حدودی است). متد Add به این شکل کار میکند که کلید را به تمام $hashfunction$ ها میدهد و تعداد عدد دریافت میکند بین صفر تا اندازه فیلتر و بیت های متناظر با این اعداد را برابر یک قرار میدهد. متد $Test$ هم به این شکل کار میکند که مجدد کلید را به تمام $hashfunction$ ها میدهد و اعداد متناظر را دریافت میکند، اگر بیت های متناظر با تمام این اعداد یک بودند جواب مثبت و در غیر این صورت منفی بر می گرداند.

حالا که با $BloomFilter$ آشنا شدید، می خواهیم با آن یک مساله واقعی حل کنیم. مساله این است که آیا پسورد شما لو رفته؟ تا به حال نزدیک پانصد میلیون پسورد واقعی از کاربران لو رفته که میتوانید در این سایت آن ها را مشاهده کنید. کل پسورد ها هم ده گیگابایت هستند که می توانید دانلودشان کنید. فرض کنیم شما می خواهید یک اپلیکیشن موبایل بنویسید که بدون دسترسی به شبکه(سایت بالا) بتواند چک کند که آیا پسورد شما داخل این لیست هست یا نه. برای این کار میتوانید از $BloomFilter$ استفاده کنید. در اینصورت اگر به کاربر گفتید که پسوردت لو نرفته که حتما نرفته ولی اگر گفتین لو رفته، خوب با یه احتمالی لو رفته(بسته به اندازه فیلتر شما). در این سوال شما باید سازنده و متدهای $Test$ و Add را برای کلاس $Q4BloomFilter$ پیاده سازی کنید بطوری که تست کلاس $Q4BloomFilterTests$ پاس شود. برای پاس شدن این تست لازم است که $FalsePositiveRate$ ده درصد کمتر باشد و اندازه فیلتر هم از پنج درصد اندازه کل پسورد کوچکتر باشد. این مقداری را در یونیت تست با یک میلیون پسورد تصادفی تست می کنیم. توی کلاس یک خانواده از $hashfunction$ ها را برای کلید های از نوع $string$ یاد گرفتیم. کار اصلی شما این است که در سازنده از این خانواده به تعداد لازم تابع انتخاب کنید و در متد های $Test$ و Add از آنها استفاده کنید. برای فهم بهتر میتوانید از این سایت استفاده کنید.

```
// Bloom filter: https://www.jasondavies.com/bloomfilter/
using System;
using System.Collections;
using System.Linq;
using TestCommon;


namespace A10
{
    public class Q4BloomFilter
    {
        BitArray Filter;
        Func<string, int>[] HashFunctions;

        public Q4BloomFilter(int filterSize, int hashFnCount)
        {
          // Write your code here to Initialize 'Filter' and 'HashFunctions' ...
        }


        public void Add(string str)
        {
            for (int i=0; i< HashFunctions.Length; i++)
            {
                Filter[HashFunctions[i](str)] = true;
            }
        }
```

```
        public bool Test(string str)
        {
            for (int i=0; i<HashFunctions.Length; i++)
            {
                if (Filter[HashFunctions[i](str)] == true)
                {
                    continue;
                }
                return false;
            }
            return true;
        }
    }
}
```